NONLINEAR OPTIMIZATION

Qingsha Cheng 程庆沙

Stochastic Search Methods I

Motivation

Random search

Simulated annealing

Introduction to population-based search

Evolution strategies

Particle swarm optimization

Stochastic Search Methods: Motivation

Gradient-based methods are **local** and **require derivative information** to proceed

Derivatives may not be available or too expensive to evaluate

Objective function may not be differentiable nor even continuous

Objective function may have multiple local minima

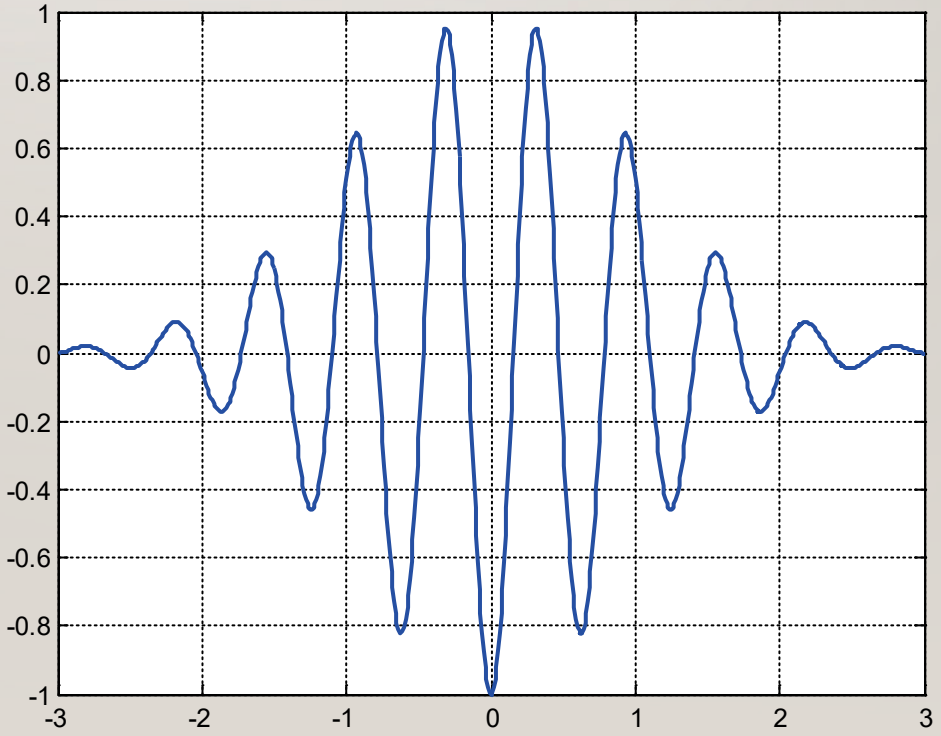
We may be interested in a comprehensive information about the system at the presence of multiple and competing objectives

Stochastic Search Methods: Motivation

Example: consider a function $f_p(x)$ defined as

$$f_p(x) = -\exp\left(-\frac{\| x \|^2}{2}\right)\prod_{j=1}^{n}\cos\left(10x_j\right)$$

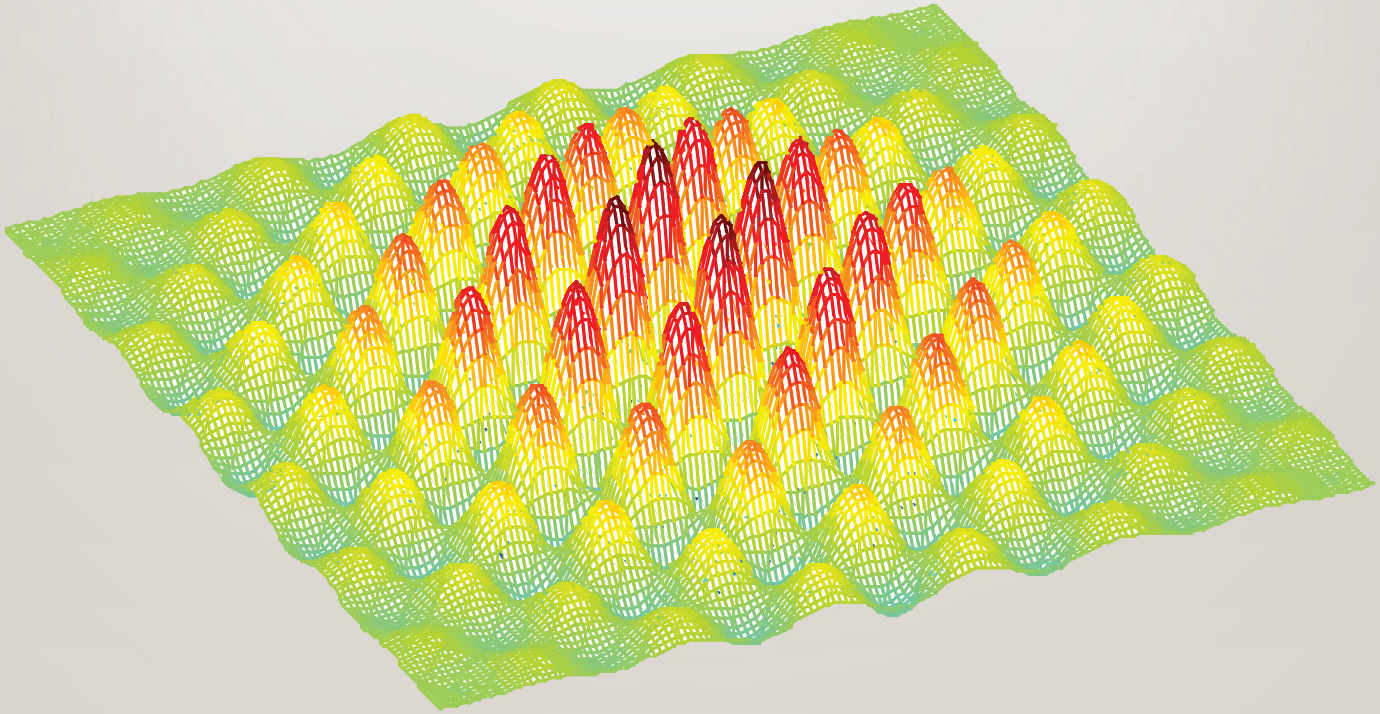
Plot of $f_p(x)$ for $n = 1$:



this function is "deceptive"--- can be fooled by unlucky sampling

Stochastic Search Methods: Motivation

Plot of $f_p(x)$ for $n = 2$:

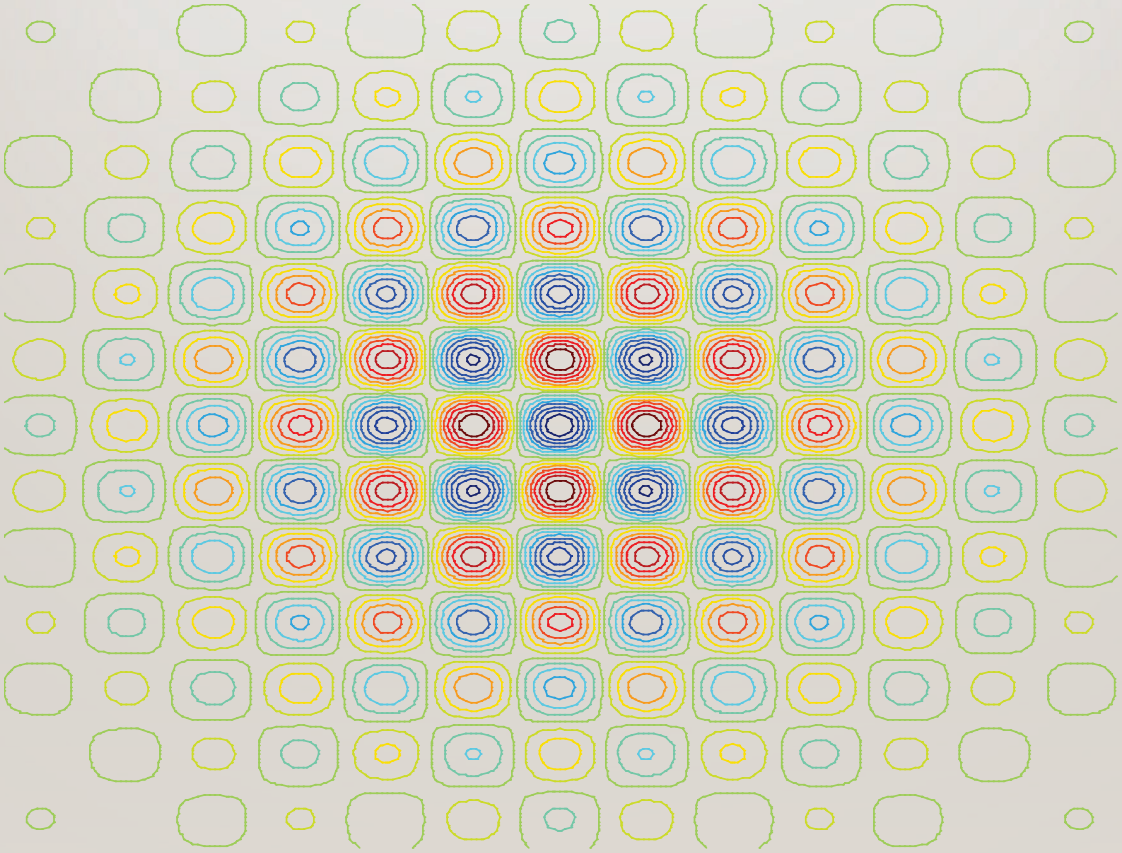


Function f_p has infinite number of local minima and maxima!

Global minimum at $[0\ 0]^T$ (function value equals -1)

Stochastic Search Methods: Motivation

Contour plots of $f_p(x)$ for $n = 2$:



Escaping from Local Minimum: Random-Start Gradient Search

One of the methods frequently used in practice in order to avoid being trapped in a local minimum is multiple-run gradient-based search using random starting points:

f_{best} = Inf;
k = 0;
while $k < k_{max}$
    x_{new} = generate_starting_point();
    [x_{new},f_{new}] = perform_gradient_search(x_{new});
    if $f_{new} < f_{best}$
        $x_{best} = x_{new}$;
        $f_{best} = f_{new}$;
    end
    $k = k + 1$;
end

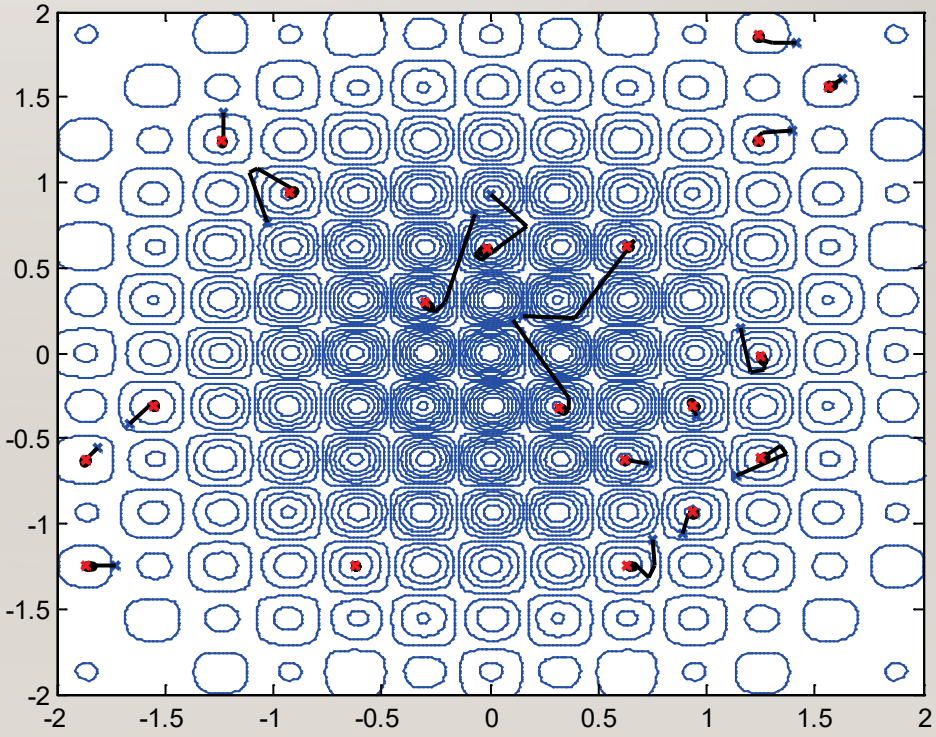
Example: Random-Start Gradient Search Minimizing f_p

Setup:

1. Local optimization routine: quasi-Newton algorithm
2. 20 local optimizations (random starting point) performed
3. in each local optimization, the number of function evaluations restricted to 50

Minimum function value found: -0.901

Total number of function evaluations: ~ 1000



Example: Random-Start Gradient Search Minimizing f_p

Statistics of 20 runs of the multiple-run gradient-based search method minimizing the function f_p

Best Result	Average Result	Worst Result	Standard Deviation
-0.9997	-0.8457	-0.6099	0.129

Remark: this method is not very reliable for the considered problem; note that there is 85 local minima (only one global) in the region of interest [-2 2] × [-2 2], so even with 20 random starting points it is difficult to get close to the global minimum

Escaping from Local Minimum: Random Search

Another method that can be used to avoid being trapped in a local minimum is a pure random search (sampling the search space with a uniform distribution):

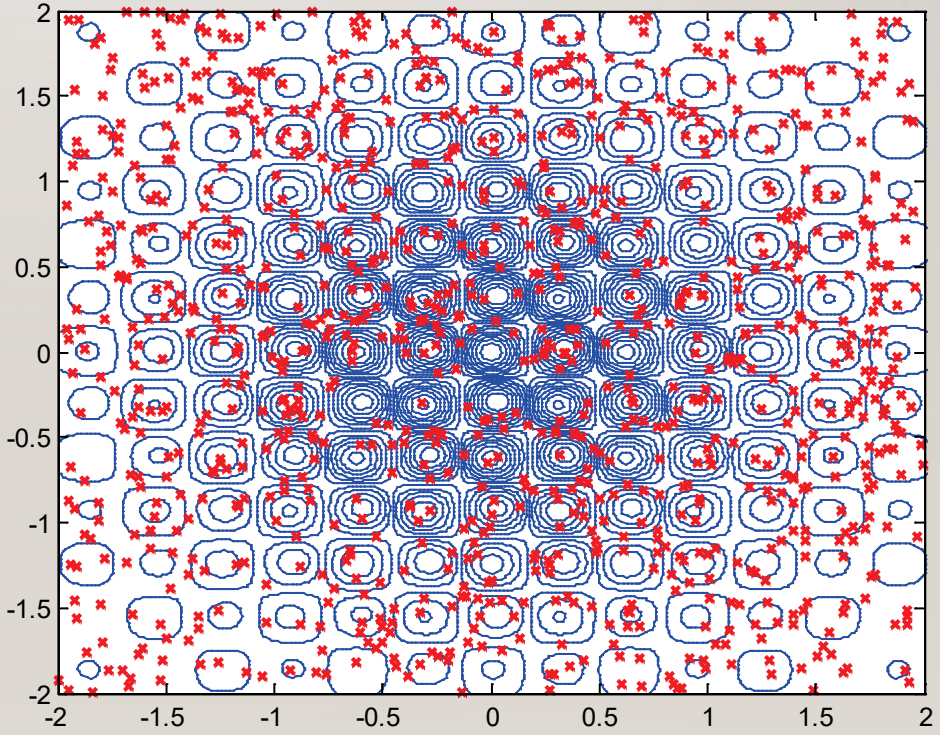
f_{best} = Inf;
k = 0;
while $k < k_{max}$
    x_{new} = generate_random_point();
    $f_{new} = f(x_{new})$;
    if $f_{new} < f_{best}$
        $x_{best} = x_{new}$;
        $f_{best} = f_{new}$;
    end
    $k = k + 1$;
end

Example: Random Search Minimizing f_p

Setup: 1000 random samples with uniform probability distribution are allocated in the search space.

Minimum function value found: -0.8896

Total number of function evaluations: 1000



Example: Random Search Minimizing f_p

Statistics of 20 runs of the random search method minimizing the function f_p

Best Result	Average Result	Worst Result	Standard Deviation
-0.9948	-0.9028	-0.7852	0.060

Remark: random search seems to work better for our problem than multiple-run gradient search; relatively large number of samples allows us to allocate at least some of them in neighborhood of the global minimum

Escaping from Local Minimum: "Smart" Random Search

Random search can be modified in many ways in order to improve its exploitation capabilities

For example, random samples can be biased by the best solution found so far, so that

$$x_{new} = \lambda x + (1 - \lambda) x_{best}$$

with $\lambda \to 0$ for $k \to k_{max}$ (maximum number of function evaluation)

This way, the search becomes more and more local in the neighborhood of x_{best} in later stages of the algorithm run

Escaping from Local Minimum: "Smart" Random Search

Algorithm:

x_{best} = generate_random_point();
$f_{best} = f(x_{best})$;
$k = 0$;
while $k < k_{max}$
    x_{new} = generate_random_point();
    $x_{new} = \lambda x_{new} + (1-\lambda)x_{best}$;
    $f_{new} = f(x_{new})$;
    if $f_{new} < f_{best}$
        $x_{best} = x_{new}$;
        $f_{best} = f_{new}$;
    end
    λ = update_lambda(k,k_{max});
    $k = k + 1$;
end

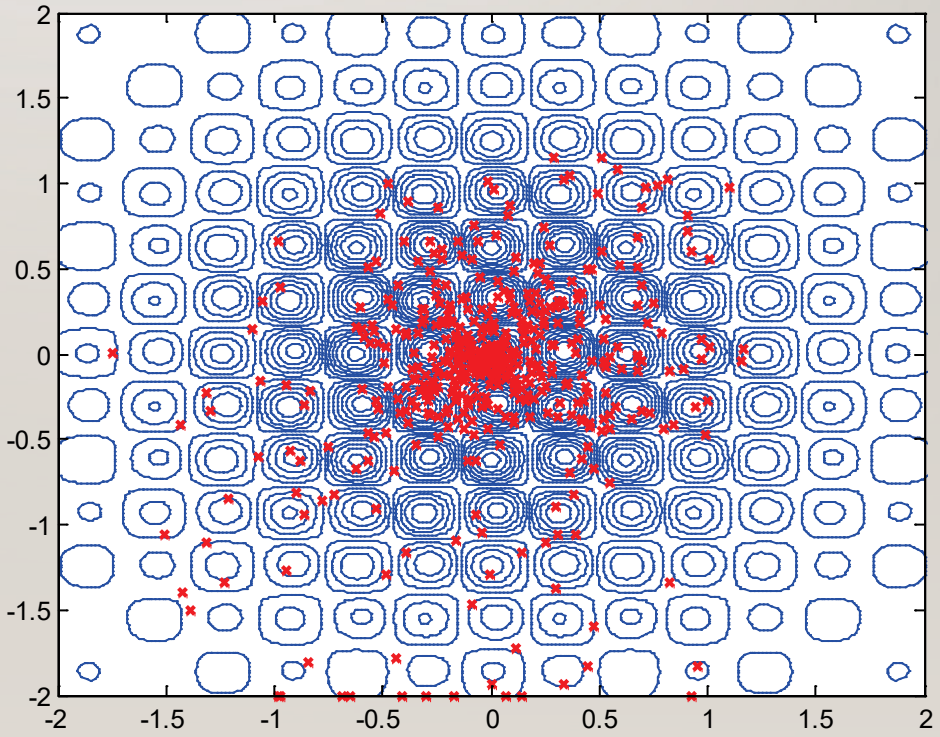
Example: "Smart" Random Search Minimizing f_p

Setup:

1. $k_{max} = 1000$.
2. Updating scheme: $\lambda = \lambda_0(1\text{-}k/k_{max})$, with $\lambda_0 = 0.5$.

Minimum function value found: -0.9996

Total number of function evaluations: 1000



Example: "Smart" Random Search Minimizing f_p

Statistics of 20 runs of the "smart" random search minimizing the function f_p

Best Result	Average Result	Worst Result	Standard Deviation
-1.0000	-0.9951	-0.9061	0.021

Remark: "smart" random search seems to be a good method for the considered problem; repeatability of the results is very good (average result close to the best result)

Simulated Annealing

annealing is a heat treatment that alters the microstructure of a material causing changes in properties such as strength, hardness, and ductility

video

Simulated Annealing

a generic **probabilistic** algorithm popular in 1980's, but has its roots in 1950's

Simulated annealing algorithm replaces the current solution by a random "nearby" solution, chosen with a **probability** that depends on the **difference** between the **corresponding function values** and on a **global parameter** *T* (called temperature) that gradually decreases during the process

Normally, the current solution changes almost **randomly** when *T* is **large**, and increasingly "**downhill**" when *T* goes to **zero**; **possibility of moving "uphill" reduces the chance of being trapped in a local minimum**

Simulated Annealing

Algorithm:

$x_{best} = x = x^{(0)}$; $f_{best} = f = f(x)$;
$k = 0$;
while $k < k_{max}$
    $x_{new} = \text{neighbor}(x)$;
    $f_{new} = f(x_{new})$;
    if $f_{new} < f_{best}$
        $x_{best} = x_{new}$; $f_{best} = f_{new}$;
    end
    if $P(f, f_{new}, \text{temp}(k/k_{max})) > \text{rand}()$ %%check transition proba.
        $x = x_{new}$; $f = f_{new}$;
    end
end

Simulated Annealing

Details:

1.Transition probability: popular formula is the following

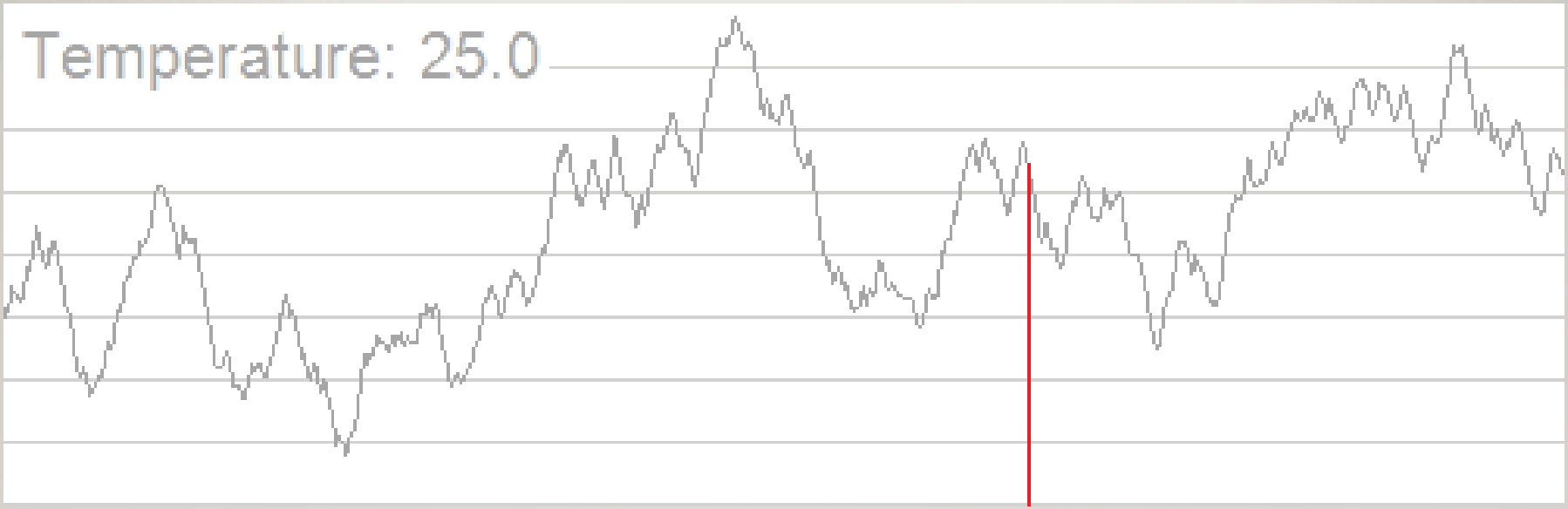
$$P(e, e_{new}, T) = \begin{cases} 1 & \text{if } e_{new} < e \\ \exp((e_{new} - e)/T) & \text{otherwise} \end{cases}$$

2.Cooling scheme: usually problem dependent and difficult to determine beforehand; in general, initial T is high and decreases to zero for $k \to k_{max}$.

3.Candidate generation: should prefer candidates with the function value similar to the one at the current state.

*Accept if e_{new} is smaller; may accept larger "uphill" step at higher temp.

Simulated Annealing



Simulated annealing searching for a maximum. The objective here is to get to the highest point; however, it is not enough to use a simple hill climb algorithm, as there are many local maxima. By cooling the temperature slowly the global maximum is found.

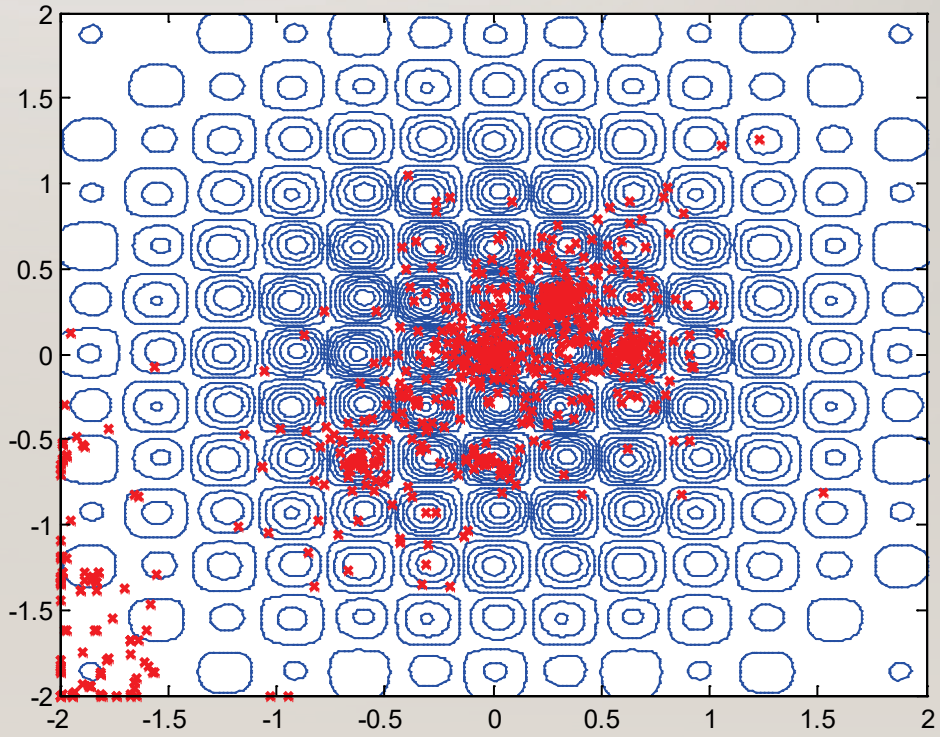
Example: Simulated Annealing Minimizing f_p

Setup:

1. P as in the previous slide
2. Cooling scheme: $T = T_0 \cdot (1\text{-}k/k_{max})$ with $T_0 = 0.1$
3. Generating neighbours: $x_{new} = x + a \cdot [r_1\ r_2]^T$, $r_i = (2 \cdot \text{rand}() - 1)$, $a = \text{rand}()^3$, $i = 1, 2$

Minimum function value found: -0.9994

Total number of function evaluations: 1000



Example: Simulated Annealing Minimizing f_p

Statistics of 20 runs of the simulated annealing algorithm minimizing the function f_p

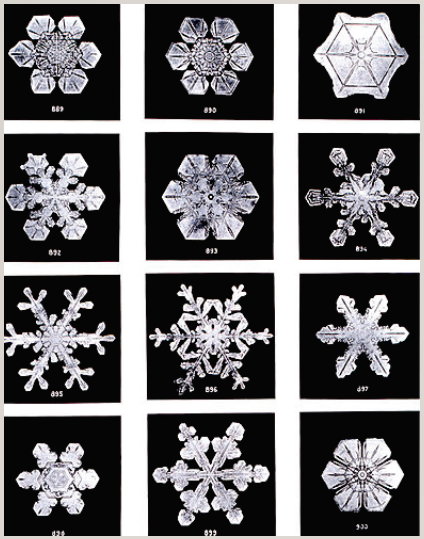
Best Result	Average Result	Worst Result	Standard Deviation
-1.0000	-0.9710	-0.9061	0.043

Remark: simulated annealing works very well with f_p; in fact, the worst result corresponds to the second-best local minimum in terms of the objective function value; repeatability of the results is also very good

Population-Based Search: Introduction

also referred to as **multi-agent systems** take inspiration from some natural systems involving groups of interacting individuals

It has been demonstrated that, as a result of these interactions, population-based algorithms exhibit **complex emergent behaviors**, like the ability to avoid getting stuck in local optima



Snowflakes

Termite mound

Population-Based Search: Introduction

have been around since late 1960s

proved to be very useful in solving many difficult problems in numerous fields of engineering and science

simulation of **complex emergent behaviors**

Population-Based Search: Terminology

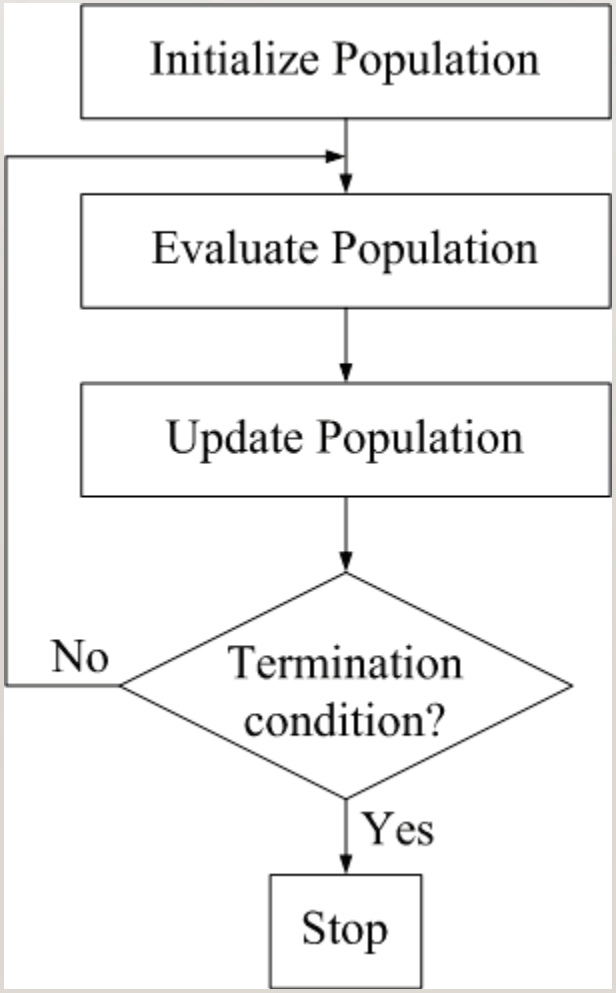
Population:	A set of individuals processed in each iteration of the algorithm
Individual:	A member of a population representing (encoded) potential solution to the problem in question
Representation:	A format used to encode the solution data
Genotype:	Encoded data of the individual
Phenotype:	Decoded data of the individual
Fitness:	A number related to the value of the objective function, determining how well is the individual doing in the population

Population-Based Search: Terminology

Evaluation:	The process of assigning fitness values to individuals
Parents:	Individuals used to create a new individual
Child, offspring:	A newly created individual
Recombination:	Process of creating new individuals
Selection:	Process of choosing parents used in recombination
Crossover:	Process of mixing parents data when creating an offspring
Mutation:	Random modification of an individual

Population-Based Search: Introduction

General flow diagram of a population-based search algorithm:



Population-Based Search: Introduction

Most popular population-based techniques:

1. Evolution strategies
2. Genetic algorithms (or more generally, evolutionary algorithms)
3. Particle swarm algorithms
4. Genetic programming
5. Ant systems

We will discuss first three of these approaches during this and the next lecture

Evolution Strategies Notations

Only if the mutants' fitness is at least as good as the parent ones, they becomes the parent of the next generation. Otherwise the mutant is disregarded.

$$(\mu + \lambda)\text{-ES}$$

If the best mutants become the parents of the next generation while the current parent is always disregarded.

$$(\mu , \lambda)\text{-ES}$$

Evolution Strategies

the **first population-based search methods** developed in mid 1960s in Germany by I. Rechenberg and H.P. Schwefel and applied for continuous optimization

process a population of μ individuals so that, in each iteration, λ offsprings are created from the set of parent individuals using problem-specific recombination operators

The new set of parent individuals is chosen either from offsprings (so-called (μ,λ)-ES) or from offsprings and parents (so-called $(\mu+\lambda)$-ES)

Each individual contains both the **solution data** (design parameters $\boldsymbol{x}$) and the set of **strategy parameters** $\boldsymbol{s}$ that determine its further mutation process

Evolution Strategies

Algorithm:

initialize population $P_\mu = \{(x^{(1)},s^{(1)}),\ldots,(x^{(\mu)},s^{(\mu)})\}$;
while ~termination_condition
  select a parent population $P_\lambda = \{(x^{(1)},s^{(1)}),\ldots,(x^{(\lambda)},s^{(\lambda)})\}$ from P_μ;
  for $j = 1{:}\lambda$
    mutate $s^{(j)}$;
    mutate $x^{(j)}$ using updated value of $s^{(j)}$;
  end
  select a new population P_μ from P_λ ((μ,λ)-selection) or from
    $P_\mu \cup P_\lambda$ ($(\mu+\lambda)$-selection);
end

Remarks:

1.Parent population is selected randomly

2.New population is chosen using deterministic truncation selection based on objective function values of individuals

Evolution Strategies

Mutation operators for unconstrained real-valued search space R^n:

$$(x,s) \leftarrow \begin{cases} s \leftarrow s \cdot e^{\tau N(0,1)} \\ x \leftarrow x + s \cdot N(0,I) \end{cases}$$

where $N(0,1)$ and $N(0,I)$ are (0,1) normally distributed random scalars and vectors, respectively, and τ is the rate of self-adaptation (learning parameters)

Parameter s controls the strength of the design parameter mutation

It is suggested that τ is proportional to $n^{1/2}$

Variations: the identity matrix I in $N(0,I)$ is sometimes replaced by a general covariance matrix with evolving entries

Evolution Strategies

Evolution strategies with **recombination**:

initialize population $P_\mu = \{(x^{(1)},s^{(1)}),\ldots,(x^{(\mu)},s^{(\mu)})\}$;
while ~termination_condition
  for $j = 1{:}\lambda$
    select a parent set $P_\rho = \{(y^{(1)},r^{(1)}),\ldots,(y^{(\rho)},r^{(\rho)})\}$ from P_μ;
    create $(x^{(j)},s^{(j)})$ by recombining $(y^{(1)},r^{(1)})$ to $(y^{(\rho)},r^{(\rho)})$;
    mutate $s^{(j)}$;
    mutate $x^{(j)}$ using updated value of $s^{(j)}$;
  end
  select a new population P_μ from P_λ ((μ,λ)-selection) or from
    $P_\mu \cup P_\lambda$ ($(\mu+\lambda)$-selection);
end

Evolution Strategies

Typical **recombination** operators:

1. Arithmetical averaging

$$x = \frac{1}{\rho} \sum_{j=1}^{\rho} y^{(j)}$$

2. Coordinate-wise random transfer

$$x = [x_1 \ldots x_n]^T = \left[y_1^{(r_1)} \ldots y_n^{(r_n)} \right]^T$$

where r_1 to r_n are randomly selected from the set $\{1,2,\ldots,\rho\}$

Evolution Strategies

Recommendations:

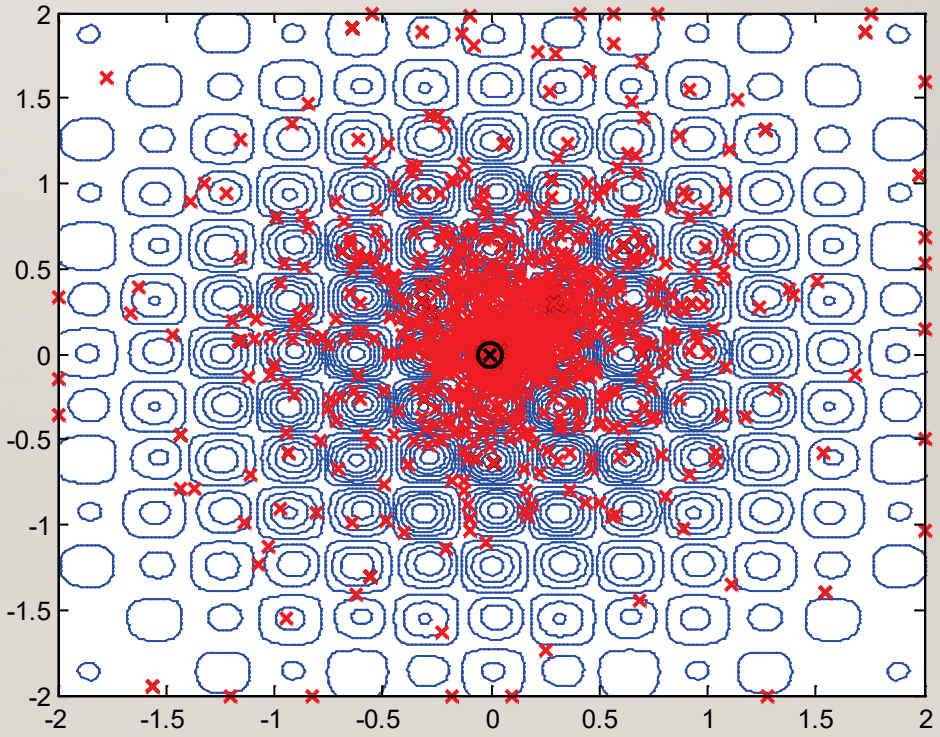
1. Typical μ/λ ratios in (μ,λ)-selection in continuous search spaces are from 1/7 to 1/2.
2. Using $(\mu+\lambda)$-selection in conjunction with variation operators (allowing reaching any point of the search space) guarantee stochastic convergence to a global solution.
3. Evolution is usually performed on a phenotypic level (i.e., no solution encoding).
4. Mutation should respect reachability condition (each point of the search space should be reachable in a finite number of steps)
5. Recombination is applied whenever possible and useful. The main goal of recombination is the conservation of common components of the parents.

Example: Evolution Strategies for Minimizing f_p

Setup:
1. Randomly chosen initial population
2. (10+10)-selection
3. Normally distributed mutation with $\tau = 0.5$,
4. Recombination: arithmetical averaging with $\rho = 2$

Minimum function value found: -0.9993

Total number of function evaluations: 1000



Example: Evolution Strategies Minimizing f_p

Statistics of 20 runs of the evolution strategies algorithm minimizing the function f_p

Best Result	Average Result	Worst Result	Standard Deviation
-1.0000	-0.9957	-0.9522	0.011

Remark: evolution strategies is a very good method for minimizing f_p; the worst result corresponds to the second-best local minimum in terms of the objective function value (as for simulated annealing); repeatability of the results is excellent

Particle Swarm Optimization

a stochastic, population-based evolutionary algorithm first introduced in 1995 by J. Kennedy (social psychologist) and R.C. Eberhart (electrical engineer)

simulates a set of individuals that interact with each other using social influence and social learning

The swarm is typically modeled by particles in a multidimensional space that have a **position** and a **velocity**. Individual particles communicate good positions to each other and adjust their own position based on these good positions

Particle Swarm Optimization

Algorithm (x_i, v_i Î R^n are position and velocity of ith particle):

initialize x_i and v_i for all i = 1,...,N;
each particle stores its (local) best $x_{best.i}$ and the swarm stores the global best g = argmin{x_i : $f(x_i)$};
while ~termination_condition
  for j = 1:N
    create random numbers $r^{(1)}$, $r^{(2)}$ from U[0,1];
    update particle velocities: v_i ¬ $c(v_i + c_1 r^{(1)} \cdot (x_{best.i} - x_i) + c_2 r^{(2)} \cdot (g - x_i))$;
    update particle positions: x_i ¬ $x_i + v_{i+1}$;
    update local bests: if $f(x_i) < f(x_{best.i})$ then $x_{best.i}$ ¬ x_i;
    update global best: if $f(x_i) < f(g)$ then g ¬ x_i;
  end
end

Remarks: 1. U[0,1] is a uniform distribution in the interval [0,1]
2. · is a component-wise multiplication
3. combination of current v and velocities to local and global best

Particle Swarm Optimization

Control parameters:

χ is typically equal to 0.7298; c_1 and c_2 say how much the particle is directed towards good positions (they represent a "cognitive" and a "social" components); usually, c_1, c_2 » 2; more precisely, for the above value of χ, c_1 and c_2 must sum to 4.1

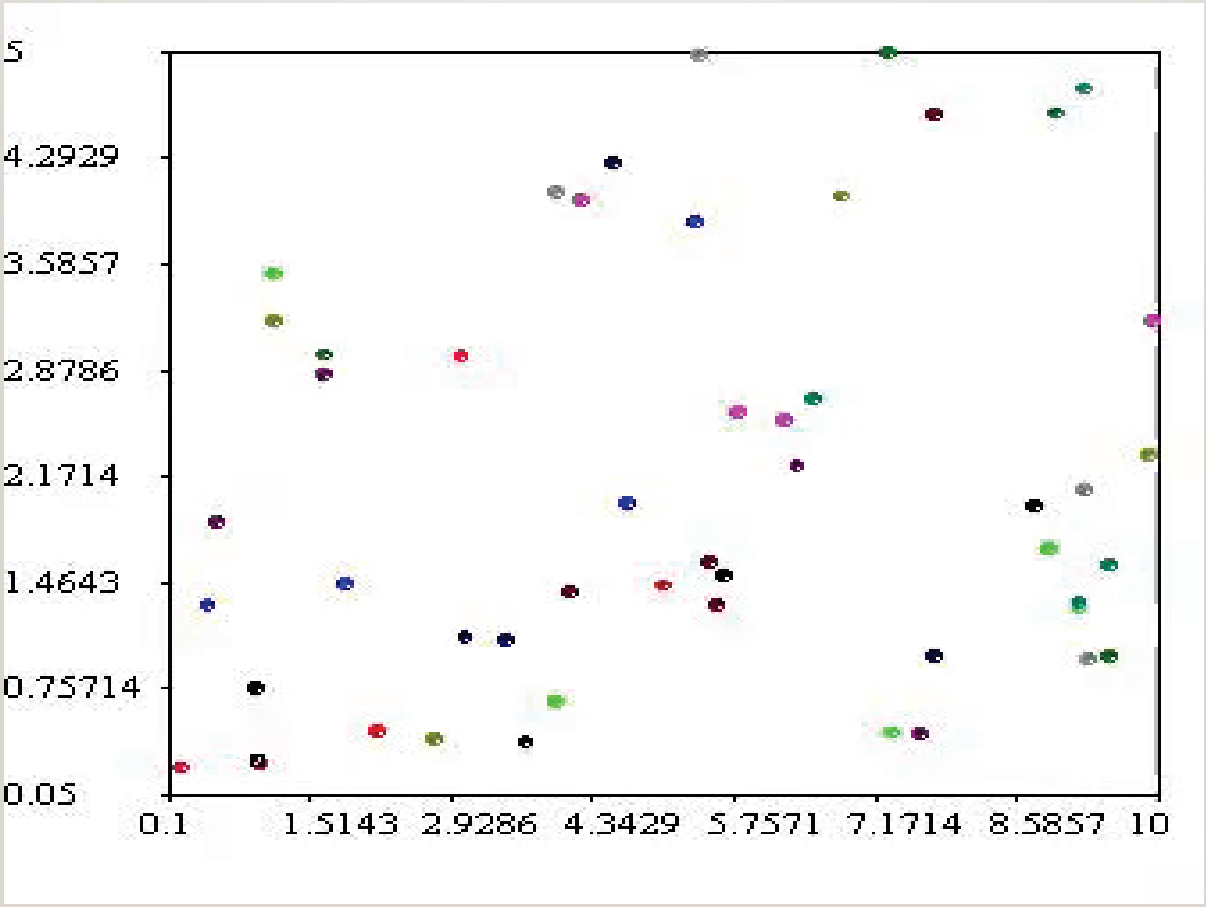
Sum of cs should not be smaller than 4.0; as cs increase, χ gets smaller as does the damping effect

The above numbers follow from the spectral analysis of the particle swarm as a dynamical system; we need to avoid "explosion" of particles and try to obtain convergence to a local (global) solution

χ may decrease over time in order to facilitate exploitation over exploration in later stages of the search

Particle Swarm Optimization

Visualization



Particle Swarm Optimization

Variations and applications:

1. **Fully informed** particle swarm: updating of the particle's velocity is based not only on the local/global best but also on the best location of a subset of the particle's neighbors; different topologies of neighbor set are considered
2. **Gradient-baset** particle swarm optimization: particle's position and velocity is updated not only based on the previous position/velocity but also on the local gradient of the objective function
3. **Major applications** of particle swarm optimization so far have been in training of artificial neural networks and finite element model updating

Example: Particle Swarm Optimization for Minimizing f_p

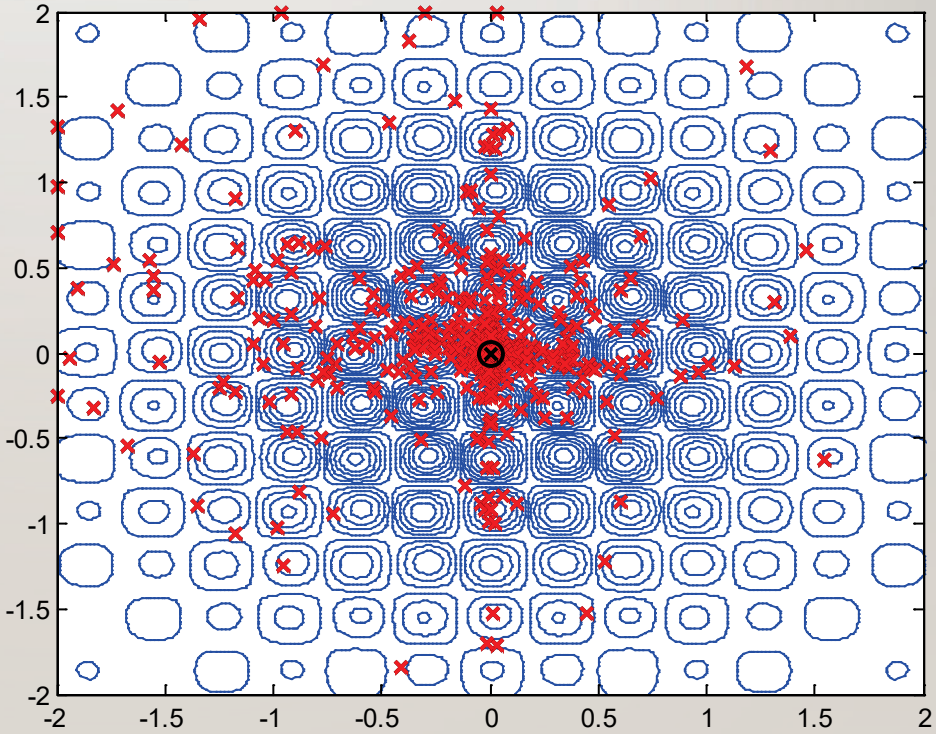
Setup: $c = 0.7298$

$c_1 = 2.05$

$c_2 = 2.05$

Minimum function value found: -0.99999

Total number of function evaluations: 1000



Example: Particle Swarm Optimization for Minimizing f_p

Statistics of 20 runs of the particle swarm optimization algorithm minimizing the function f_p

Best Result	Average Result	Worst Result	Standard Deviation
-1.0000	-0.9538	-0.8255	0.055

Remark: particle swarm optimization is a good method for minimizing f_p, however, it is not as good as simulated annealing or evolution strategies.

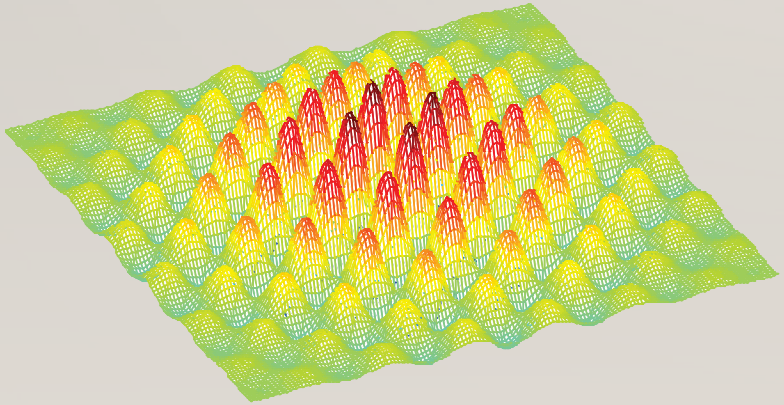
Comparison of Stochastic Search Methods: Test Function 1

Test function 1: $$f_p(x) = -\exp\left(-\frac{\|x\|^2}{2}\right)\prod_{j=1}^{n}\cos\left(10x_j\right)$$

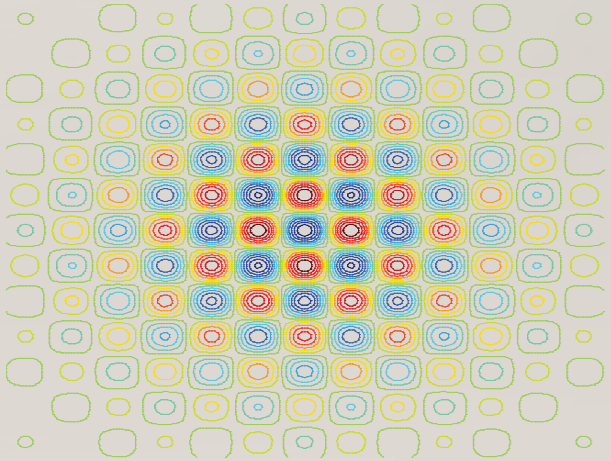
Test case: $n = 2$, search space: $[-2\ 2] \times [2\ 2]$

Global minimum -1 at $[0\ 0]^T$

Surface plot

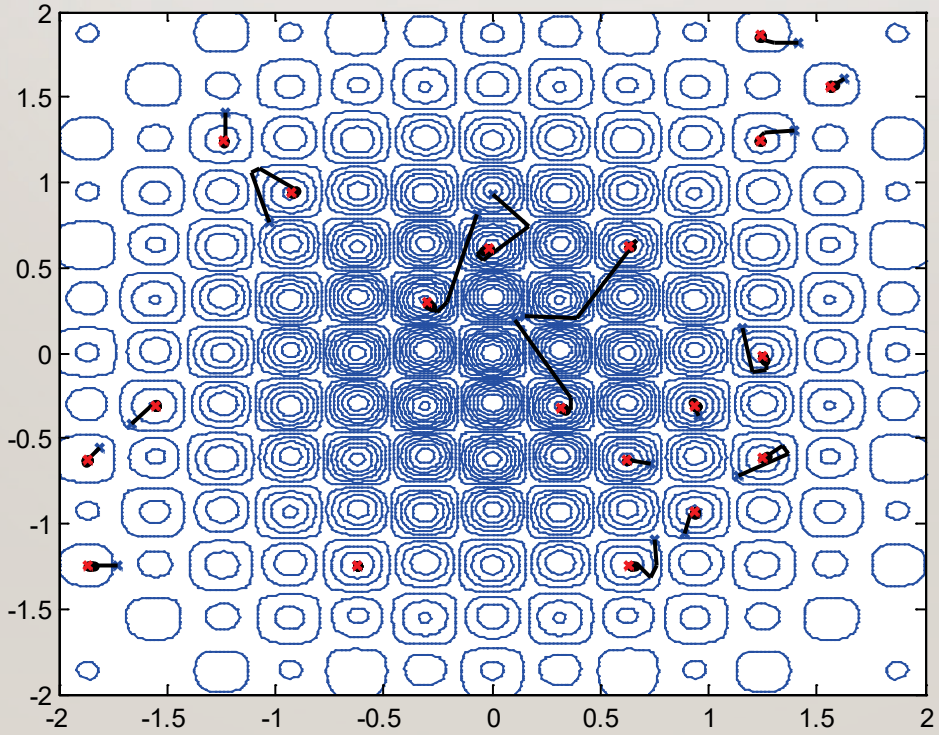


Contour plot



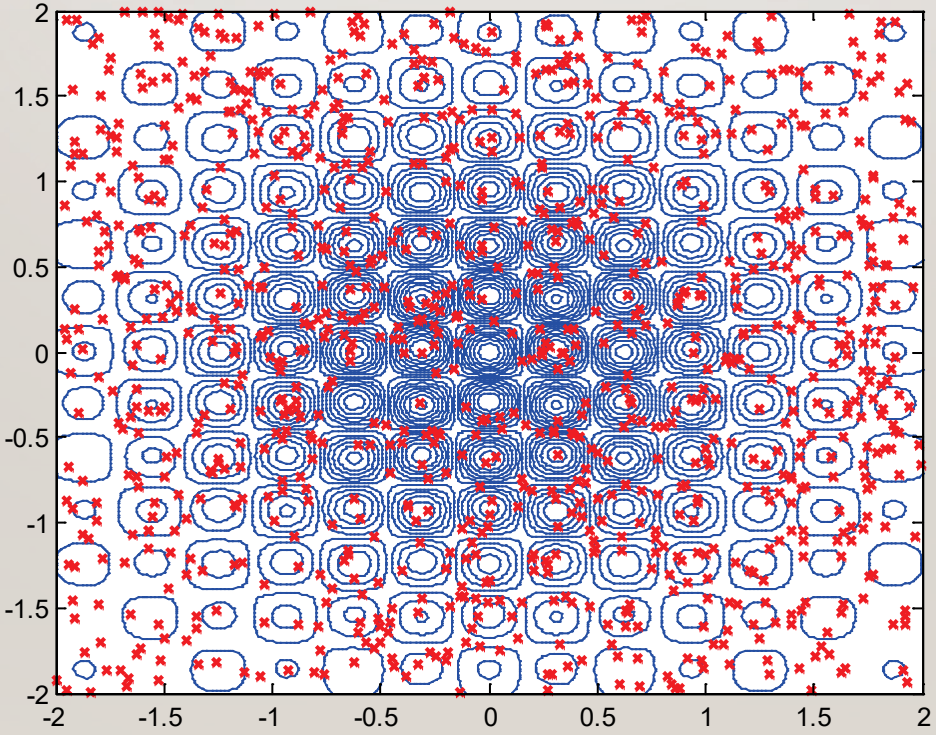
Comparison of Stochastic Search Methods: Test Function 1

Typical search pattern for multiple-start gradient-based method:



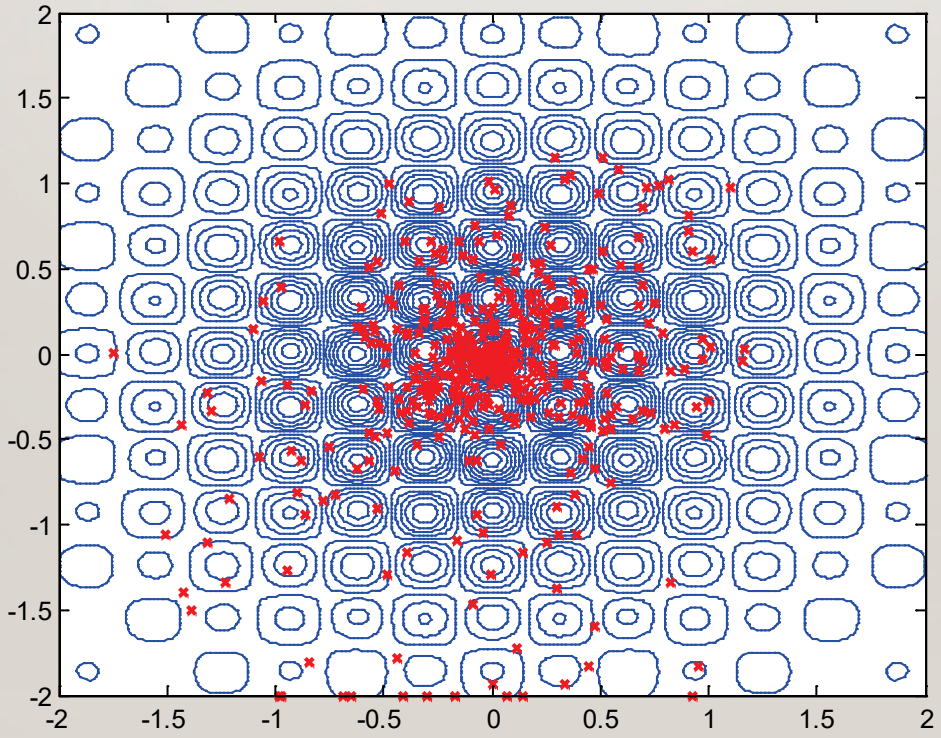
Comparison of Stochastic Search Methods: Test Function 1

Typical search pattern for random search method:



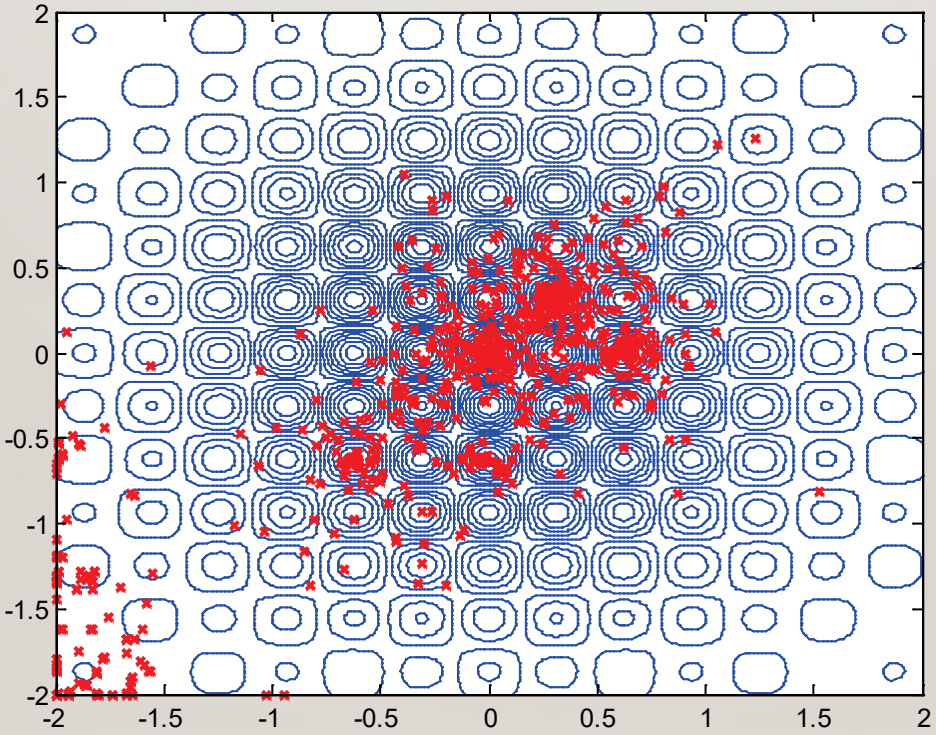
Comparison of Stochastic Search Methods: Test Function 1

Typical search pattern for “smart” random search method:



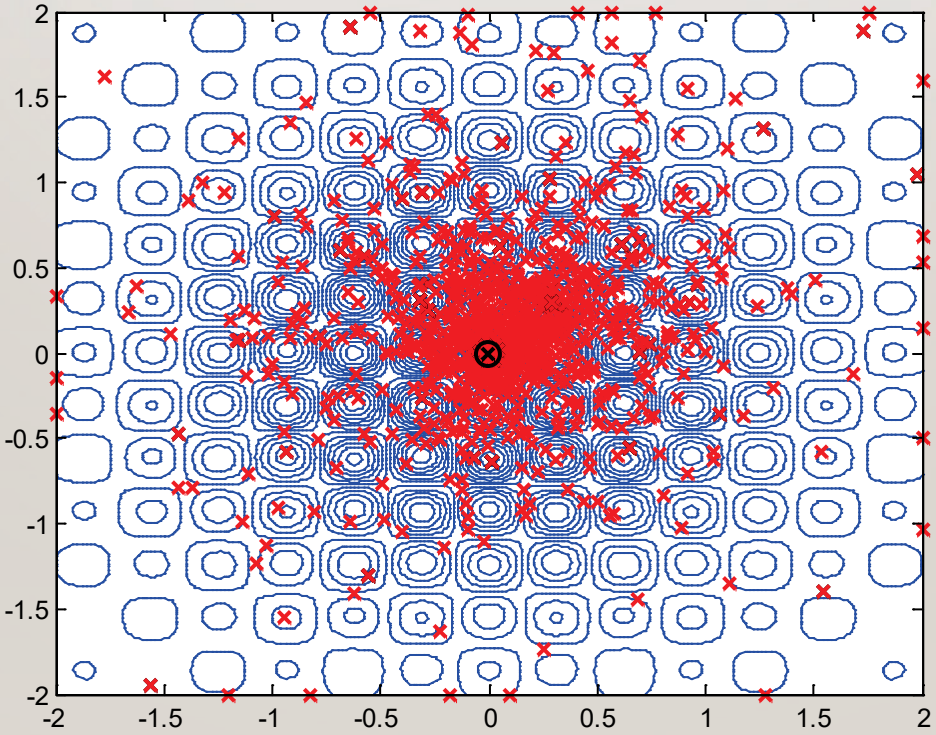
Comparison of Stochastic Search Methods: Test Function 1

Typical search pattern for simulated annealing algorithm:



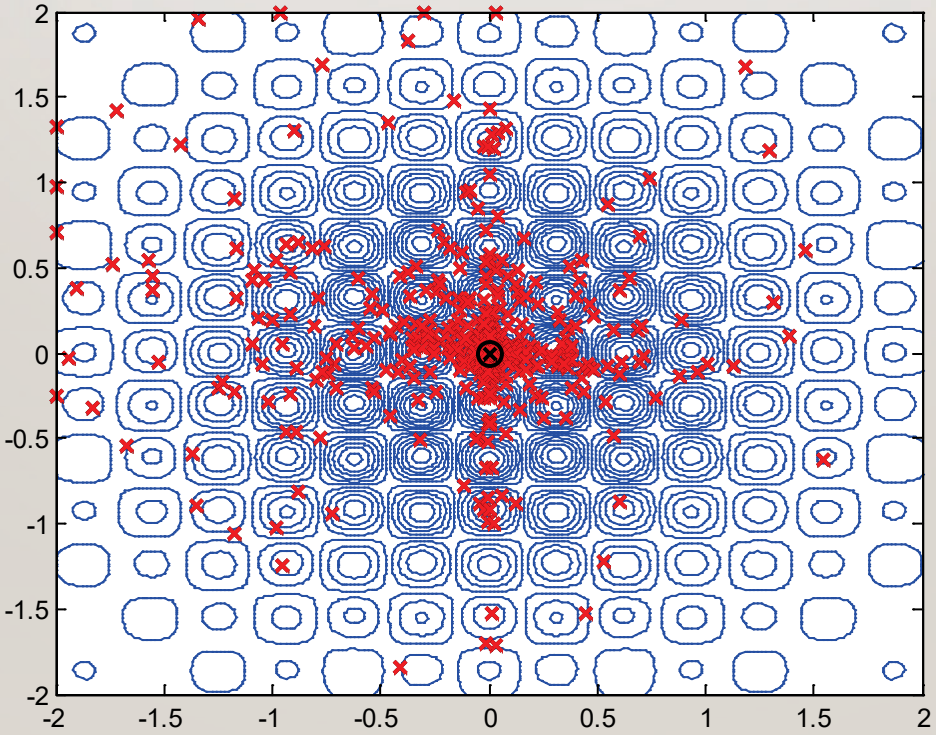
Comparison of Stochastic Search Methods: Test Function 1

Typical search pattern for evolution strategies algorithm:



Comparison of Stochastic Search Methods: Test Function 1

Typical search pattern for particle swarm optimization algorithm:



Comparison of Stochastic Search Methods:| Test Function 1

Statistics based on 20 runs of each algorithm

Algorithm	Best Result	Average Result	Worst Result	Standard Deviation
Multiple-start gradient-based search	-0.9997	-0.8457	-0.6099	0.129
Random search	-0.9948	-0.9028	-0.7852	0.060
"Smart" random search	-1.0000	-0.9951	-0.9061	0.021
Simulated annealing	-1.0000	-0.9710	-0.9061	0.043
Evolution strategies	-1.0000	-0.9957	-0.9522	0.011
Particle swarm optimization	-1.0000	-0.9538	-0.8255	0.055

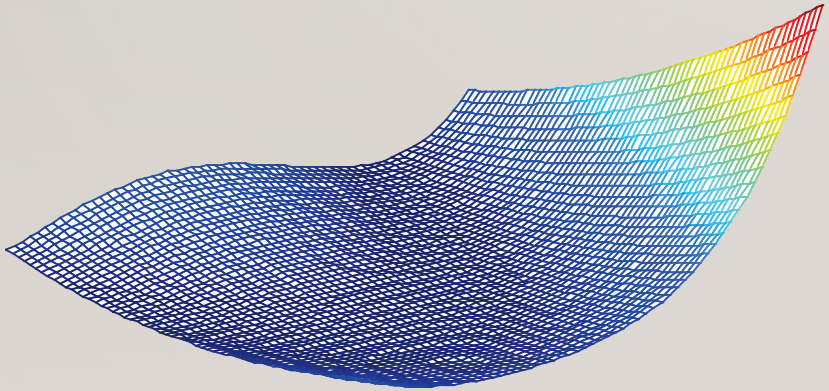
Comparison of Stochastic Search Methods: Test Function 2

Test function 2 (Rosenbrock): $$f_r(x) = \sum_{i=1}^{n-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2 \right]$$

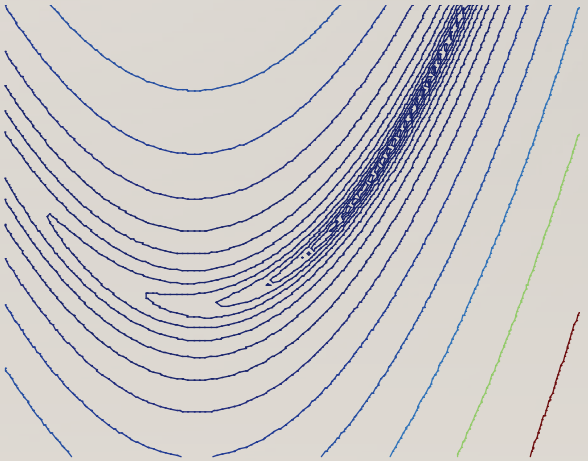
Test case: $n = 2$, search space: [-1 -1] × [2 2]

Global minimum 0 at $[1\ 1]^T$

Surface plot

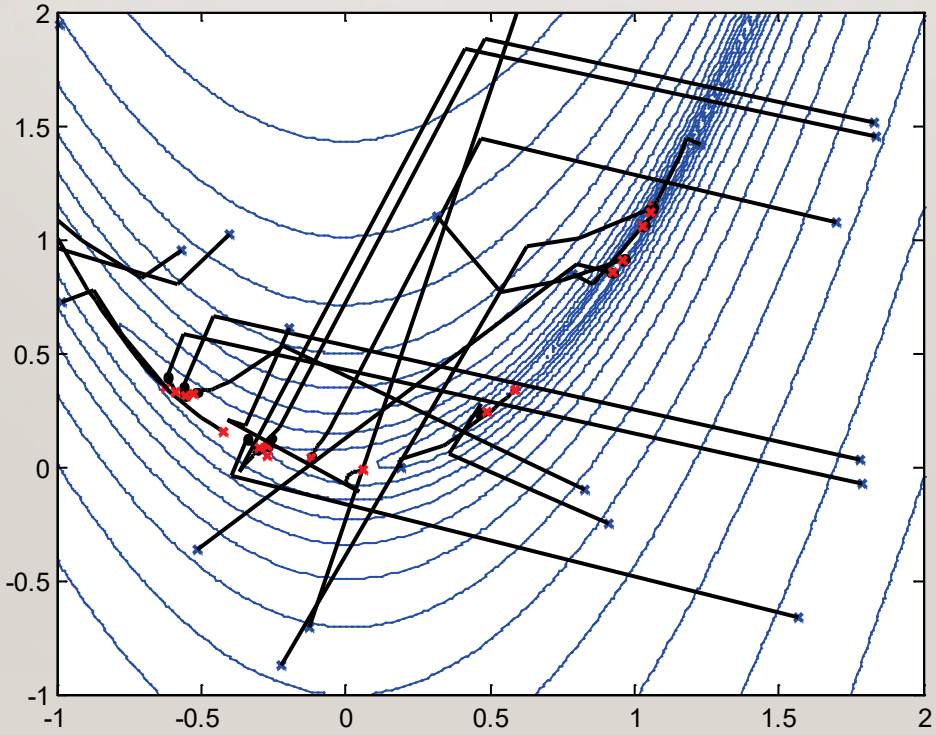


Contour plot



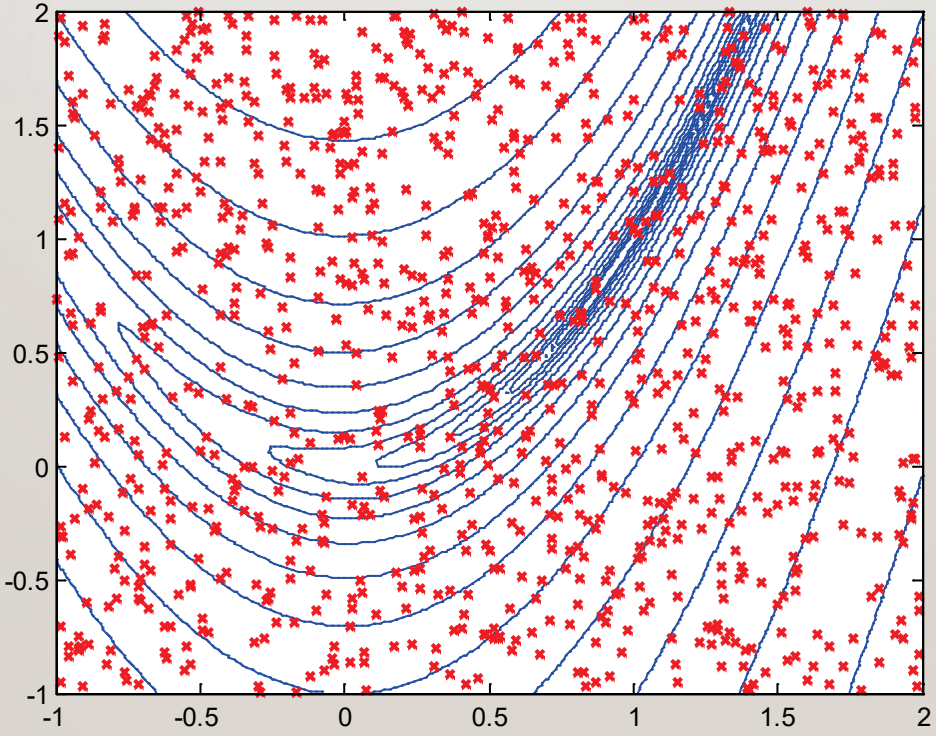
Comparison of Stochastic Search Methods: Test Function 2

Typical search pattern for multiple-start gradient-based method:



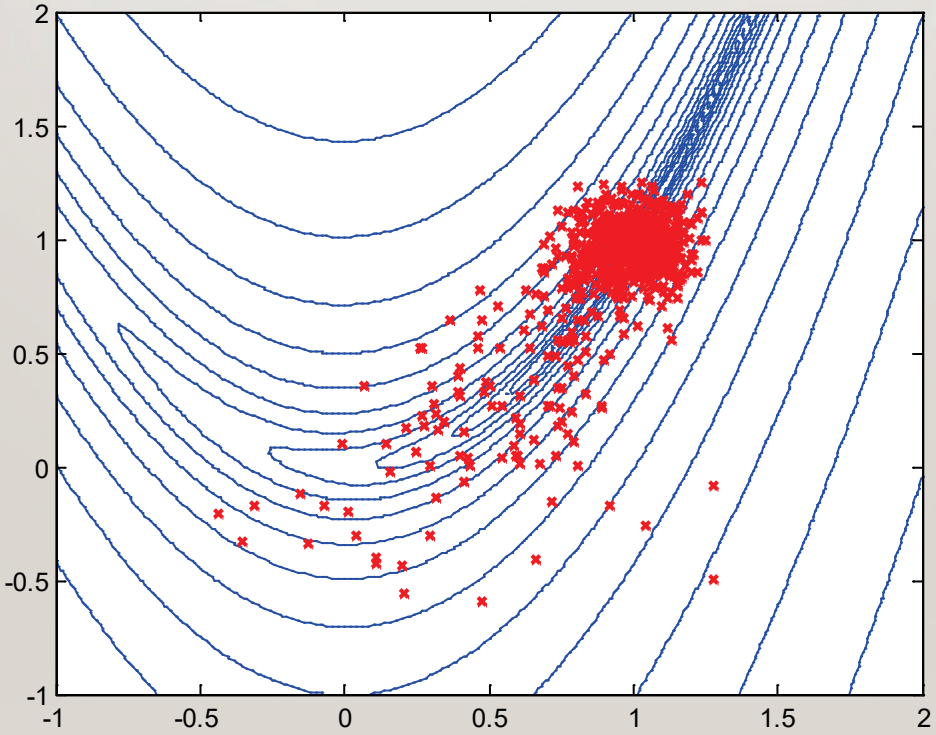
Comparison of Stochastic Search Methods: Test Function 2

Typical search pattern for random search method:



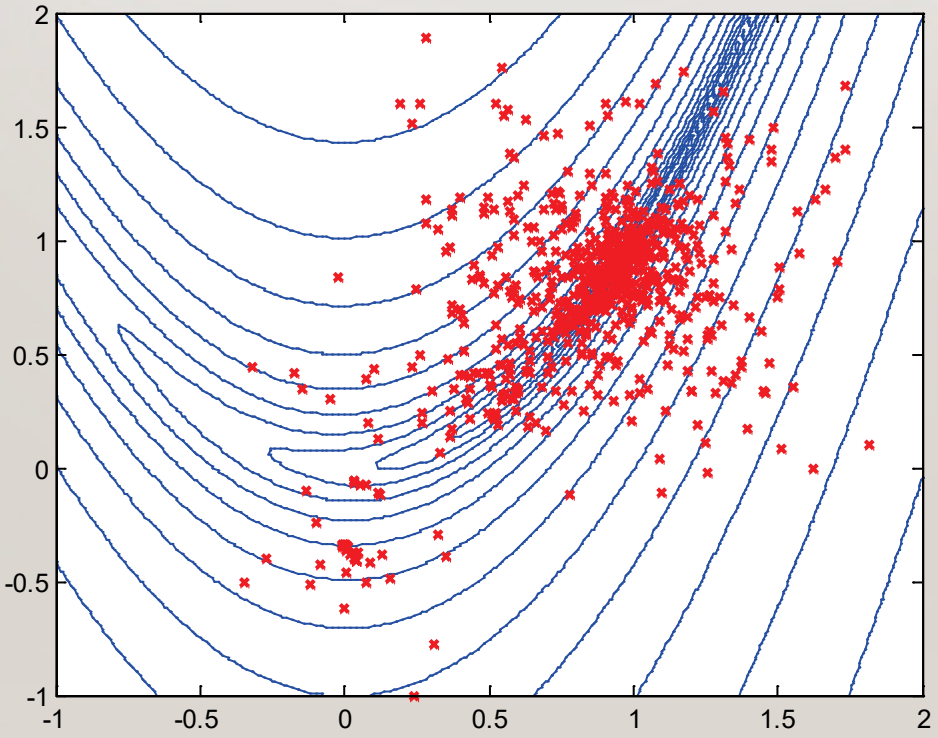
Comparison of Stochastic Search Methods: Test Function 2

Typical search pattern for “smart” random search method:



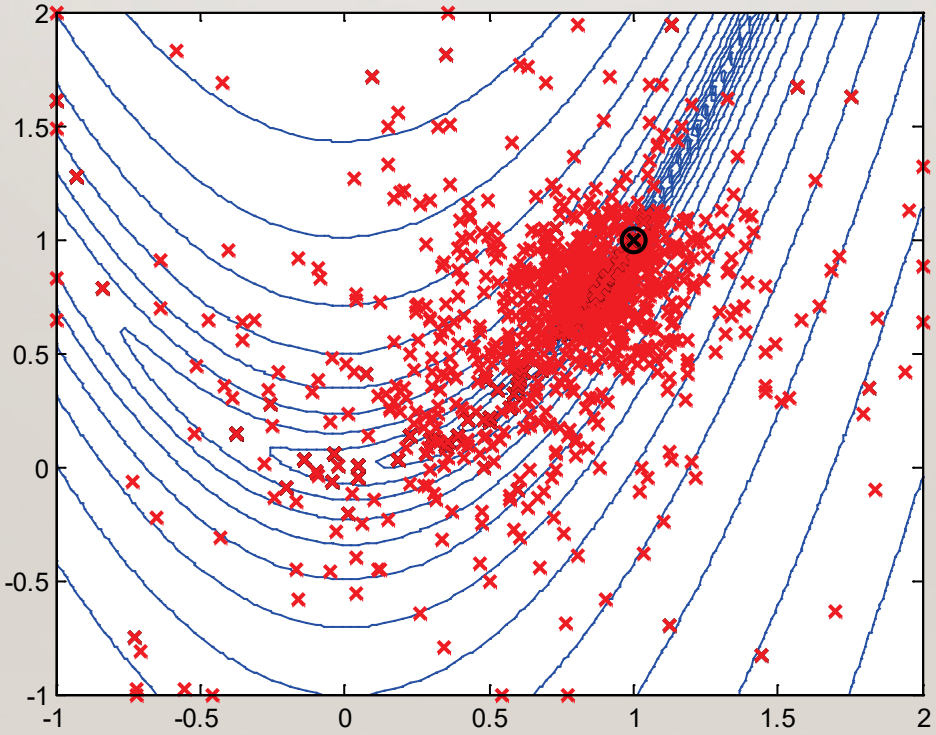
Comparison of Stochastic Search Methods: Test Function 2

Typical search pattern for simulated annealing algorithm:



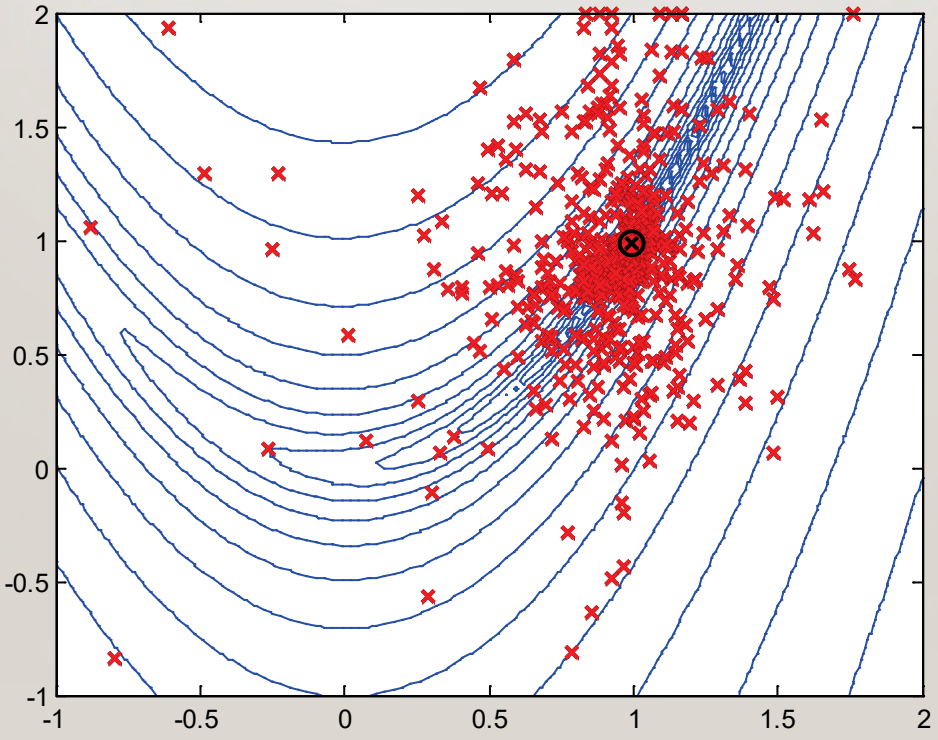
Comparison of Stochastic Search Methods: Test Function 2

Typical search pattern for evolution strategies algorithm:



Comparison of Stochastic Search Methods: Test Function 2

Typical search pattern for particle swarm optimization algorithm:



Comparison of Stochastic Search Methods: Test Function 2

Statistics based on 20 runs of each algorithm

Algorithm	Best Result	Average Result	Worst Result	Standard Deviation
Multiple-start gradient-based search*	$2.4 \cdot 10^{-6}$ (0^*)	$8.2 \cdot 10^{-2}$ (0^*)	$3.1 \cdot 10^{-2}$ (0^*)	$1.0 \cdot 10^{-2}$ (0^*)
Random search	$4.5 \cdot 10^{-5}$	$3.7 \cdot 10^{-2}$	$9.6 \cdot 10^{-2}$	$3.3 \cdot 10^{-2}$
"Smart" random search	$6.2 \cdot 10^{-7}$	$1.5 \cdot 10^{-4}$	$1.7 \cdot 10^{-3}$	$3.9 \cdot 10^{-4}$
Simulated annealing	$5.0 \cdot 10^{-6}$	$8.5 \cdot 10^{-4}$	$5.6 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$
Evolution strategies	$3.9 \cdot 10^{-7}$	$4.7 \cdot 10^{-3}$	$4.3 \cdot 10^{-2}$	$9.7 \cdot 10^{-3}$
Particle swarm optimization	$2.8 \cdot 10^{-7}$	$5.1 \cdot 10^{-3}$	$2.3 \cdot 10^{-2}$	$7.7 \cdot 10^{-3}$

* The single-start algorithm always finds a global minimum with sufficient number of function evaluations

Comparison of Stochastic Search Methods: Test Function 3

Test function 3 (Auckley):

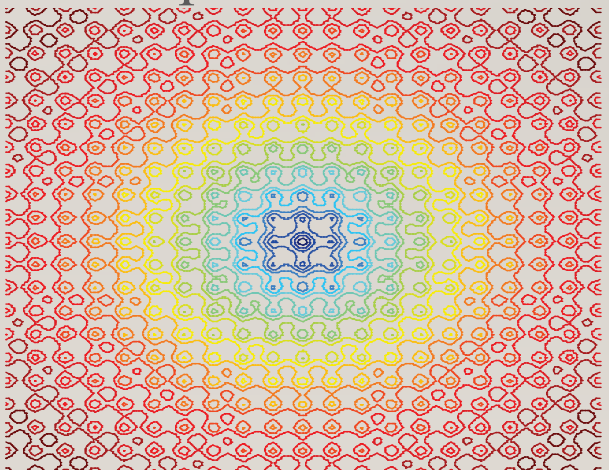
$$f_a(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 21$$

Test case: $n = 2$, search space: [-10 -10] × [10 10]

Global minimum -1.7183 at $[0\ 0]^T$

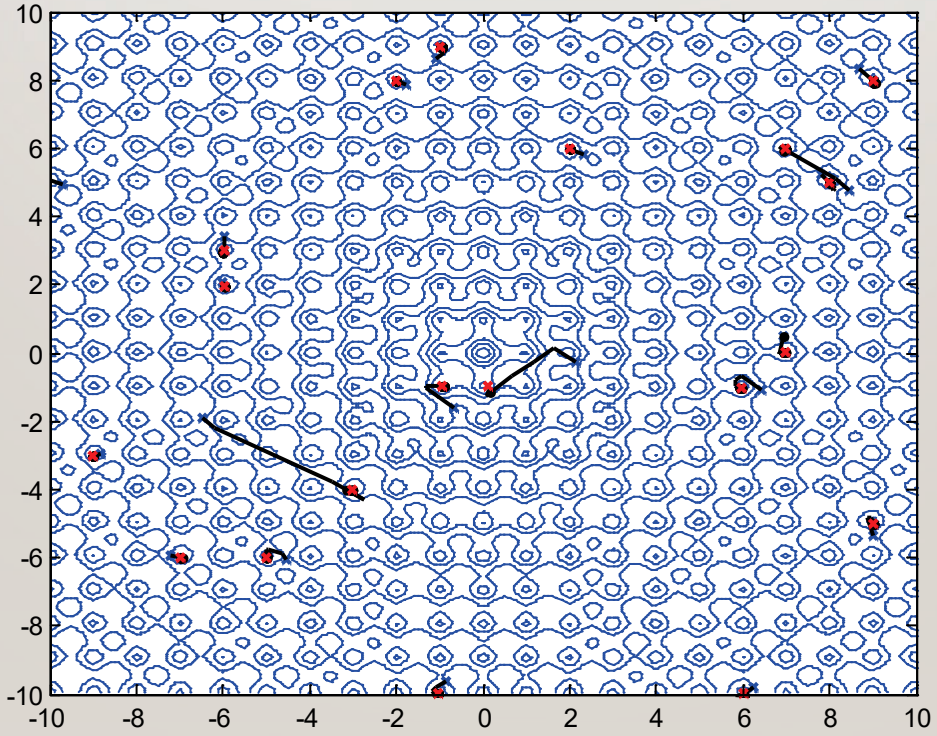
Surface plot

Contour plot



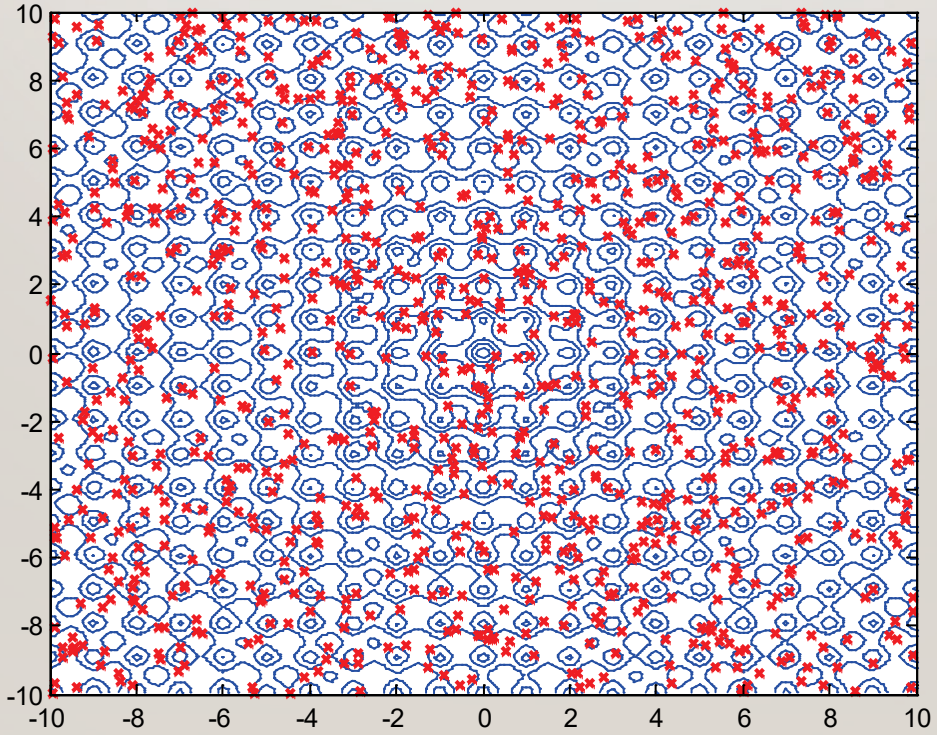
Comparison of Stochastic Search Methods: Test Function 3

Typical search pattern for multiple-start gradient-based method:



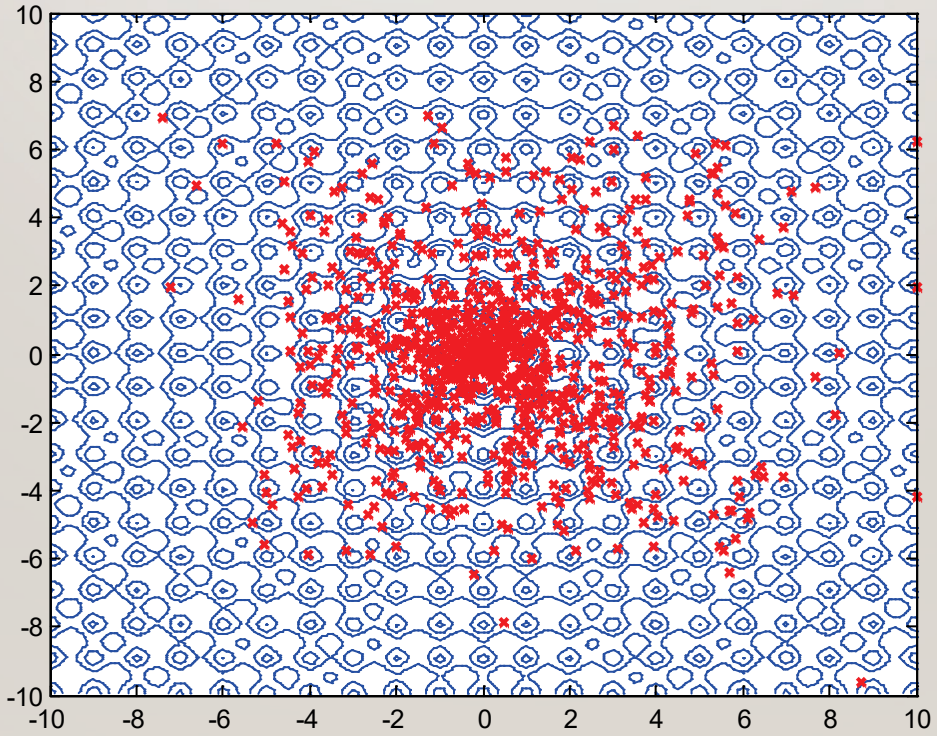
Comparison of Stochastic Search Methods: Test Function 3

Typical search pattern for random search method:



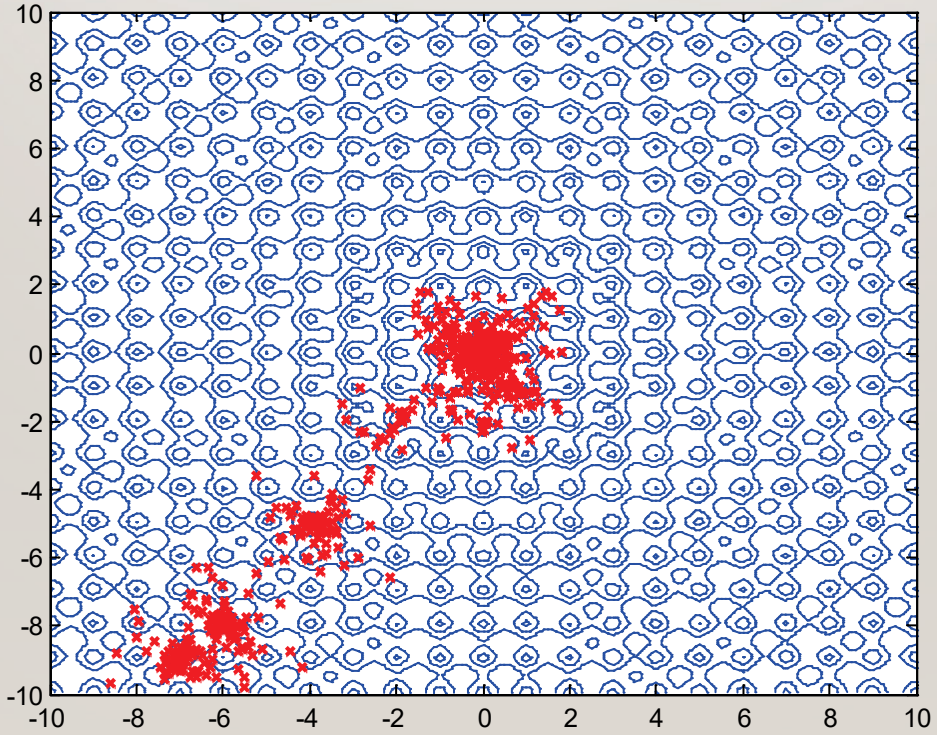
Comparison of Stochastic Search Methods: Test Function 3

Typical search pattern for “smart” random search method:



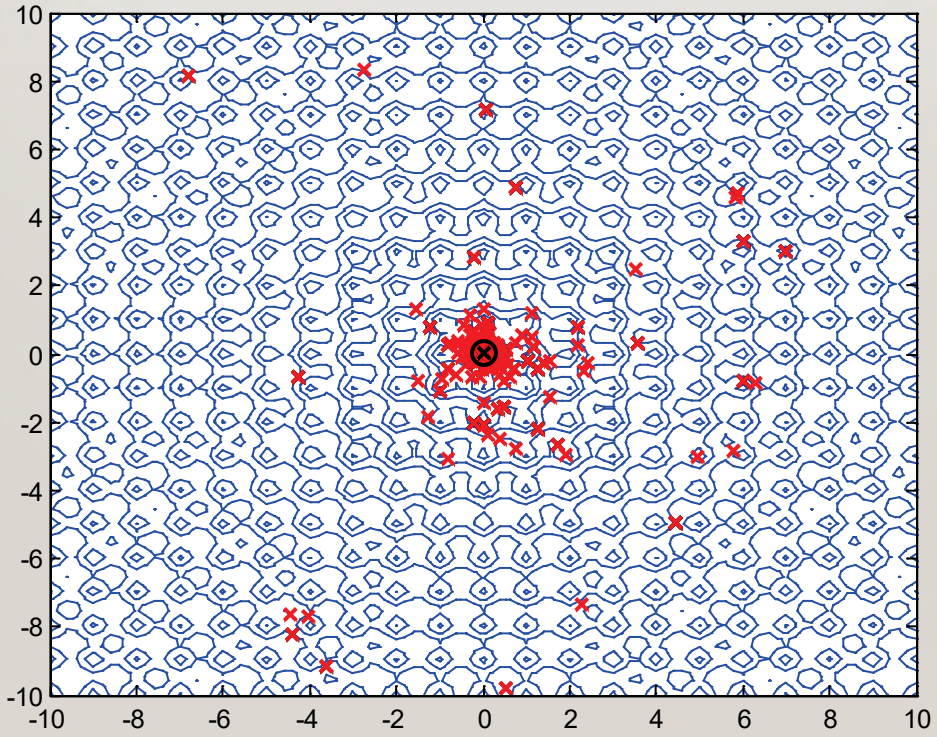
Comparison of Stochastic Search Methods: Test Function 3

Typical search pattern for simulated annealing algorithm:



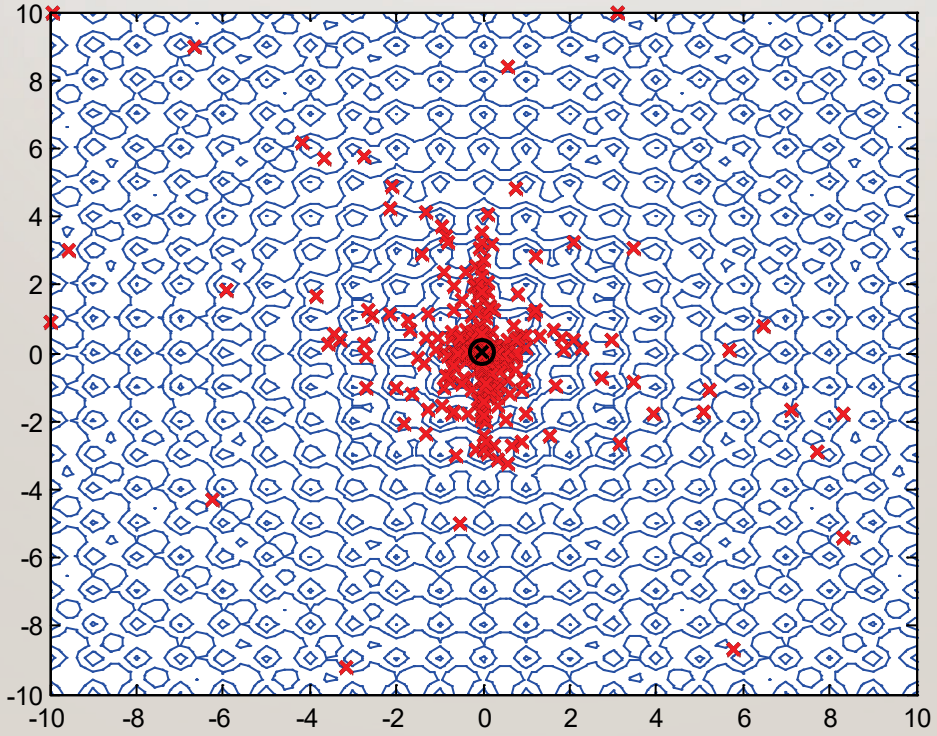
Comparison of Stochastic Search Methods: Test Function 3

Typical search pattern for evolution strategies algorithm:



Comparison of Stochastic Search Methods: Test Function 3

Typical search pattern for particle swarm optimization algorithm:



Comparison of Stochastic Search Methods: Test Function 3

Statistics based on 20 runs of each algorithm

Algorithm	Best Result	Average Result	Worst Result	Standard Deviation
Multiple-start gradient-based search	-1.7182	1.7244	5.4661	2.17
Random search	-1.4596	0.2036	1.1830	0.72
"Smart" random search	-1.7106	-1.6835	-1.6496	0.02
Simulated annealing	-1.7172	-1.7132	-1.7054	0.004
Evolution strategies	-1.7183	-1.7181	-1.7153	0.0007
Particle swarm optimization	-1.7183	-1.7182	-1.7178	0.0001

Comparison of Stochastic Search Methods: Test Function 1, $n = 5$

Statistics based on 20 runs of each algorithm

Algorithm	Best Result	Average Result	Worst Result	Standard Deviation
Multiple-start gradient-based search	-0.6135	-0.3354	-0.1055	0.143
Random search	-0.4073	-0.2859	-0.1351	0.069
"Smart" random search	-0.9056	-0.7276	-0.4138	0.146
Simulated annealing	-0.9046	-0.4298	-0.1234	0.204
Evolution strategies	-1.0000	-0.7295	-0.4150	0.131
Particle swarm optimization	-0.9069	-0.6857	-0.3764	0.176

Comparison of Stochastic Search Methods: Test Function 2, $n = 5$

Statistics based on 20 runs of each algorithm

Algorithm	Best Result	Average Result	Worst Result	Standard Deviation
Multiple-start gradient-based search*	0.0330 (0*)	0.5274 (0*)	1.4186 (0*)	0.473 (0*)
Random search	7.1031	14.8165	21.3041	3.488
"Smart" random search	0.0030	1.8780	4.5043	1.613
Simulated annealing	0.0019	2.6610	4.5100	1.686
Evolution strategies	0.6380	2.1327	4.5314	1.027
Particle swarm optimization	0.0692	2.0929	4.8601	1.765

* The single-start algorithm always finds a global minimum with sufficient number of function evaluations

Comparison of Stochastic Search Methods: Test Function 3, $n = 5$

Statistics based on 20 runs of each algorithm

Algorithm	Best Result	Average Result	Worst Result	Standard Deviation
Multiple-start gradient-based search	2.6796	6.9066	9.7872	2.045
Random search	2.3630	4.5634	6.0501	0.938
"Smart" random search	-1.6426	-0.8469	1.1754	0.959
Simulated annealing	-1.7065	-0.9635	0.6013	0.859
Evolution strategies	-1.7183	-1.6024	0.5986	0.518
Particle swarm optimization	-1.7183	-1.7183	-1.7182	0.00001

Comparison of Stochastic Search Methods: Conclusions

1. Stochastic search methods should not be used for "easy" problems (here, functions with single global optimum such as Rosenbrock function).
2. Gradient based methods are superior for smooth function with single global optimum but they are not doing well for problems with multiple local optima.
3. Stochastic methods, especially population-based search methods are doing particularly well for functions with multiple local optima and clear trend (e.g., Auckley function).
4. Random search is not particularly reliable in general but may be easily improved to be comparable to some other, more sophisticated approaches; they can even be better for "deceptive" functions (e.g., f_p).

Bibliography

S. Kirkpatrick and C. D. Gelatt and M. P. Vecchi, „Optimization by simulated annealing," Science, vol. 220, no. 4598, pp. 671-680, 1983.

T. Back, D.B. Fogel, and Z. Michalewicz (Editors), *Evolutionary computation 1: basic algorithms and operators*, Taylor & Francis Group, 2000.

H.-G. Beyer and H.-P. Schwefel, „Evolution strategies: a comprehensive introduction," *Journal Natural Computing*, 1(1):3-52, 2002.

J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks, Perth, Australia,* Nov. 1995*, pp.* 1942–1948.

J. Kennedy, "The particle swarm: Social adaptation of knowledge," in *Proc. 1997 Int. Conf. Evolutionary Computation, Indianapolis, IN,* Apr. 1997, pp. 303–308.

M. Clerc and J. Kennedy, "The particle swarm: explosion, stability, and convergence in a multi-dimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, pp. 58–73, Feb. 2002.

Exercise 1: Multiple-Run Gradient Search

Implement multiple-run gradient-search algorithm using random starting point in each run. The results should be displayed after each run including run number, best objective function value found so far, number of function evaluations.

Test the algorithm on:
(1)Rosenbrock function ($n = 2$, $-2 \leq x_i \leq 2$),
(2)Function f_p ($n = 1, 2$, $-2 \leq x_i \leq 2$),
(3)Auckley function ($n = 1, 2$ and 3, $-10 \leq x_i \leq 10$).

Exercise 2: Random Search

Implement random search algorithm using uniform probability distribution in the search space. The results should be displayed after each iteration best value of the objective function value found so far and the number of function evaluations.

Modify the algorithm to make the search “smarter”.

Test the algorithm on:
(1)Rosenbrock function ($n = 2$, $-2 \leq x_i \leq 2$),
(2)Function f_p ($n = 1, 2$, $-2 \leq x_i \leq 2$),
(3)Auckley function ($n = 1, 2$ and 3, $-10 \leq x_i \leq 10$).

Compare the basic version of the random search with the “smart” one.

Exercise 3: Differential Evolution Algorithm

Differential evolution (DE) algorithm is one of the most recent population search methods used for minimization of function f of n variables. Outline ($0 \leq F \leq 2$ and $0 \leq CR \leq 1$ are user defined control parameters):

- Initialize N individuals (random selection);
- Until termination repeat:
  - For each individual $x = [x_1 \ldots x_n]$:
    - Pick three random individuals y^1, y^2, y^3 (all, including x, distinct);
    - Pick a random index $p \in \{1, \ldots, n\}$;
    - Generate new individual $y = [y_1 \ldots y_n]$ by iterating for $i = 1$ to n:
      - Pick random $r \in (0,1)$ (uniform distribution);
      - If $i = p$ or $r < CR$, $y_i = y_i^1 + F(y_i^2 - y_i^3)$, otherwise $y_i = x_i$;
      - If $f(y) < f(x)$ set $x = y$;

Test the algorithm on:
(1)Rosenbrock function ($n = 2$, $-2 \leq x_i \leq 2$),
(2)Function f_p ($n = 1, 2$, $-2 \leq x_i \leq 2$),
(3)Auckley function ($n = 1, 2$ and 3, $-10 \leq x_i \leq 10$).
Use $F = 1$ and $CR = 0.5$ initially, then change F and CR to see how it affects the algorithm performance.

Exercise 4: Particle Swarm Optimization

Implement the particle swarm optimization algorithm.

Test the algorithm on:
(1)Rosenbrock function ($n = 2$, $-2 \leq x_i \leq 2$),
(2)Function f_p ($n = 1, 2$, $-2 \leq x_i \leq 2$),
(3)Auckley function ($n = 1, 2$ and 3, $-10 \leq x_i \leq 10$).

Perform some experiments to check how the performance of the algorithm depends on its control parameters (e.g., N, c, c_1, c_2, etc.).